Lexicographic generation. Table 1 shows combinations $a_{n-1}\ldots a_1a_0$ and $c_t\ldots c_1$ in lexicographic order, which is also the lexicographic order of $d_t\ldots d_1$. Notice that the dual combinations $b_s\ldots b_1$ and the corresponding compositions $p_t\ldots p_0$, $q_t\ldots q_0$ then appear in *reverse* lexicographic order.

Lexicographic order usually suggests the most convenient way to generate combinatorial configurations. Indeed, Algorithm 7.2.1.2L already solves the problem for combinations in the form $a_{n-1}\ldots a_1a_0$, since (s,t)-combinations in bitstring form are the same as permutations of the multiset $\{s\cdot 0, t\cdot 1\}$. That general-purpose algorithm can be streamlined in obvious ways when it is applied to this special case. (See also exercise 7.1.3–20, which presents a remarkable sequence of seven bitwise operations that will convert any given binary number $(a_{n-1}\ldots a_1a_0)_2$ to the lexicographically next t-combination, assuming that n does not exceed the computer's word length.)

Let's focus, however, on generating combinations in the other principal form $c_t\ldots c_2c_1$, which is more directly relevant to the ways in which combinations are often needed, and which is more compact than the bit strings when t is small compared to n. In the first place we should keep in mind that a simple sequence of nested loops will do the job nicely when t is very small. For example, when $t=3$ the following instructions suffice:

For $c_3 = 2, 3, \ldots, n-1$ (in this order) do the following:
For $c_2 = 1, 2, \ldots, c_3 - 1$ (in this order) do the following:
For $c_1 = 0, 1, \ldots, c_2 - 1$ (in this order) do the following: (15)
Visit the combination $c_3c_2c_1$.

(See the analogous situation in 7.2.1.1–(3).)

On the other hand when t is variable or not so small, we can generate combinations lexicographically by following the general recipe discussed after Algorithm 7.2.1.2L, namely to find the rightmost element c_j that can be increased and then to set the subsequent elements $c_{j-1}\ldots c_1$ to their smallest possible values:

Algorithm L (*Lexicographic combinations*). This algorithm generates all t-combinations $c_t\ldots c_2c_1$ of the n numbers $\{0,1,\ldots,n-1\}$, given $n \ge t \ge 0$. Additional variables c_{t+1} and c_{t+2} are used as sentinels.

L1. [Initialize.] Set $c_j \leftarrow j-1$ for $1 \le j \le t$; also set $c_{t+1} \leftarrow n$ and $c_{t+2} \leftarrow 0$.

L2. [Visit.] Visit the combination $c_t\ldots c_2c_1$.

L3. [Find j.] Set $j \leftarrow 1$. Then, while $c_j + 1 = c_{j+1}$, set $c_j \leftarrow j-1$ and $j \leftarrow j+1$; eventually the condition $c_j + 1 \ne c_{j+1}$ will occur.

L4. [Done?] Terminate the algorithm if $j > t$.

L5. [Increase c_j.] Set $c_j \leftarrow c_j + 1$ and return to L2. ∎

The running time of this algorithm is not difficult to analyze. Step L3 sets $c_j \leftarrow j-1$ just after visiting a combination for which $c_{j+1} = c_1 + j$, and the number of such combinations is the number of solutions to the inequalities

$$n > c_t > \cdots > c_{j+1} \ge j; \quad (16)$$

but this formula is equivalent to a $(t-j)$-combination of the $n-j$ objects $\{n-1,\ldots,j\}$, so the assignment $c_j \leftarrow j-1$ occurs exactly $\binom{n-j}{t-j}$ times. Summing for $1 \le j \le t$ tells us that the loop in step L3 is performed

$$\binom{n-1}{t-1}+\binom{n-2}{t-2}+\cdots+\binom{n-t}{0} = \binom{n-1}{s}+\binom{n-2}{s}+\cdots+\binom{s}{s} = \binom{n}{s+1} \qquad (17)$$

times altogether, or an average of

$$\binom{n}{s+1}\bigg/\binom{n}{t} = \frac{n!}{(s+1)!\,(t-1)!}\bigg/\frac{n!}{s!\,t!} = \frac{t}{s+1} \qquad (18)$$

times per visit. This ratio is less than 1 when $t \le s$, so Algorithm L is quite efficient in such cases.

But the quantity $t/(s+1)$ can be embarrassingly large when t is near n and s is small. Indeed, Algorithm L occasionally sets $c_j \leftarrow j-1$ needlessly, at times when c_j already equals $j-1$. Further scrutiny reveals that we need not always search for the index j that is needed in steps L4 and L5, since the correct value of j can often be predicted from the actions just taken. For example, after we have increased c_4 and reset $c_3c_2c_1$ to their starting values 210, the next combination will inevitably increase c_3. These observations lead to a tuned-up version of the algorithm:

Algorithm T (*Lexicographic combinations*). This algorithm is like Algorithm L, but faster. It also assumes, for convenience, that $t < n$.

T1. [Initialize.] Set $c_j \leftarrow j-1$ for $1 \le j \le t$; then set $c_{t+1} \leftarrow n$, $c_{t+2} \leftarrow 0$, and $j \leftarrow t$.

T2. [Visit.] (At this point j is the smallest index such that $c_{j+1} > j$.) Visit the combination $c_t \ldots c_2c_1$. Then, if $j > 0$, set $x \leftarrow j$ and go to step T6.

T3. [Easy case?] If $c_1 + 1 < c_2$, set $c_1 \leftarrow c_1 + 1$ and return to T2. Otherwise set $j \leftarrow 2$.

T4. [Find j.] Set $c_{j-1} \leftarrow j-2$ and $x \leftarrow c_j + 1$. If $x = c_{j+1}$, set $j \leftarrow j+1$ and repeat step T4.

T5. [Done?] Terminate the algorithm if $j > t$.

T6. [Increase c_j.] Set $c_j \leftarrow x$, $j \leftarrow j-1$, and return to T2. ▮

Now $j = 0$ in step T2 if and only if $c_1 > 0$, so the assignments in step T4 are never redundant. Exercise 6 carries out a complete analysis of Algorithm T.

Notice that the parameter n appears only in the initialization steps L1 and T1, not in the principal parts of Algorithms L and T. Thus we can think of the process as generating the first $\binom{n}{t}$ combinations of an *infinite* list, which depends only on t. This simplification arises because the list of t-combinations for $n+1$ things begins with the list for n things, under our conventions; we have been using lexicographic order on the decreasing sequences $c_t \ldots c_1$ for this very reason, instead of working with the increasing sequences $c_1 \ldots c_t$.

Derrick Lehmer noticed another pleasant property of Algorithms L and T [*Applied Combinatorial Mathematics*, edited by E. F. Beckenbach (1964), 27–30]:

Theorem L. *The combination $c_t \ldots c_2 c_1$ is visited after exactly*

$$\binom{c_t}{t} + \cdots + \binom{c_2}{2} + \binom{c_1}{1} \qquad (19)$$

other combinations have been visited.

Proof. There are $\binom{c_k}{k}$ combinations $c'_t \ldots c'_2 c'_1$ with $c'_j = c_j$ for $t \ge j > k$ and $c'_k < c_k$, namely $c_t \ldots c_{k+1}$ followed by the k-combinations of $\{0, \ldots, c_k - 1\}$. ∎

When $t = 3$, for example, the numbers

$$\tbinom{2}{3}+\tbinom{1}{2}+\tbinom{0}{1},\ \tbinom{3}{3}+\tbinom{1}{2}+\tbinom{0}{1},\ \tbinom{3}{3}+\tbinom{2}{2}+\tbinom{0}{1},\ \ldots,\ \tbinom{5}{3}+\tbinom{4}{2}+\tbinom{3}{1}$$

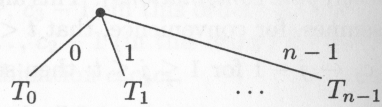
that correspond to the combinations $c_3 c_2 c_1$ in Table 1 simply run through the sequence 0, 1, 2, ..., 19. Theorem L gives us a nice way to understand the *combinatorial number system* of degree t, which represents every nonnegative integer N uniquely in the form

$$N = \binom{n_t}{t} + \cdots + \binom{n_2}{2} + \binom{n_1}{1}, \qquad n_t > \cdots > n_2 > n_1 \ge 0. \qquad (20)$$

[See Ernesto Pascal, *Giornale di Matematiche* **25** (1887), 45–49.]

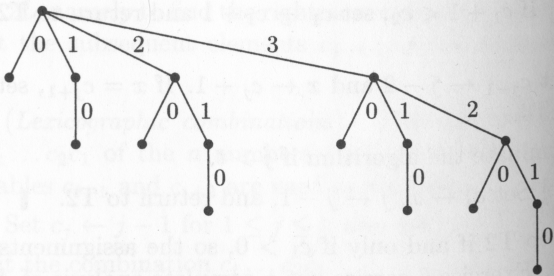
Binomial trees. The family of trees T_n defined by

$T_0 = \bullet$, $\quad T_n =$



for $n > 0$, (21)

arises in several important contexts and sheds further light on combination generation. For example, T_4 is



; (22)

and T_5, rendered more artistically, appears as the frontispiece to Volume 1 of this series of books.

Notice that T_n is like T_{n-1}, except for an additional copy of T_{n-1}; therefore T_n has 2^n nodes altogether. Furthermore, the number of nodes on level t is the binomial coefficient $\binom{n}{t}$; this fact accounts for the name "binomial tree." Indeed, the sequence of labels encountered on the path from the root to each node on level t defines a combination $c_t \ldots c_1$, and all combinations occur in lexicographic order from left to right. Thus, Algorithms L and T can be regarded as procedures to traverse the nodes on level t of the binomial tree T_n.